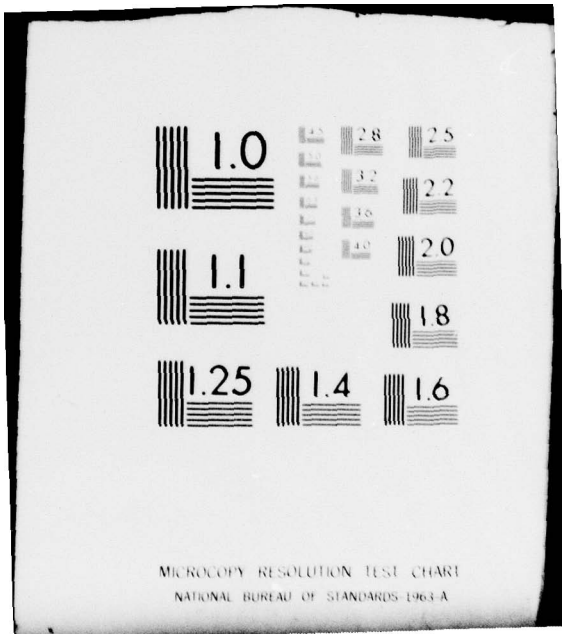
AD-A074 601 MARYLAND UNIV COLLEGE PARK DEPT OF INFORMATION SYSTE--ETC F/G 9/2
DATA MODEL PROCESSING: AN APPROACH TO STANDARDIZATION OF DATABA--ETC(U)
JUL 79 W T HARDGRAVE, E H SIBLEY DAAG29-78-G-0162
UNCLASSIFIED IFSM-TR-45 ARO-15712.3-A-EL NL

| OF |
AD
A074601

END
DATE
FILMED
10-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

ARO 15712.3-A-EL

LEVEL II

DATA MODEL PROCESSING: AN APPROACH TO STANDARDIZATION OF DATABASE MANAGEMENT SYSTEMS.

Technical rept. 1 Sep 78 - 25 Jul 79

TECHNICAL REPORT.
IFSM T.R. No. 45

by

W. Terry Hardgrave & Edgar H. Sibley

ARO

25 July ~~25,~~ 1979

15712.3-A-EL

27

Prepared for:
U.S. Army Institute for Research in Management
Information and Computer Science (AIRMICS)

under
U. S. ARMY RESEARCH OFFICE

IFSM-TR-45

GRANT NUMBER DAAG29-78-G-0162

Department of Information Systems Management
University of Maryland
College Park, MD 20742

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSIDERED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: IFSM T.R. No. 45	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): DATA MODEL PROCESSING: AN APPROACH TO STANDARDIZATION OF DATABASE MANAGEMENT SYSTEMS	5. TYPE OF REPORT & PERIOD COVERED: Technical Report, Sept. 1, 1978 - July 25, 1979
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): W. Terry Hardgrave & Edgar H. Sibley	8. CONTRACT OR GRANT NUMBER(s): DAAG29-78-G-0162
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Information Systems Management, University of Maryland, College Park, MD 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: U.S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: July 22, 1979
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official department of the Army position, policy, or decision unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DATA MODELS, POSITIONAL SET PROCESSOR, DBMS, DATA MODEL TRANSFORMATION, DEFINITION OF DATA MODELS.

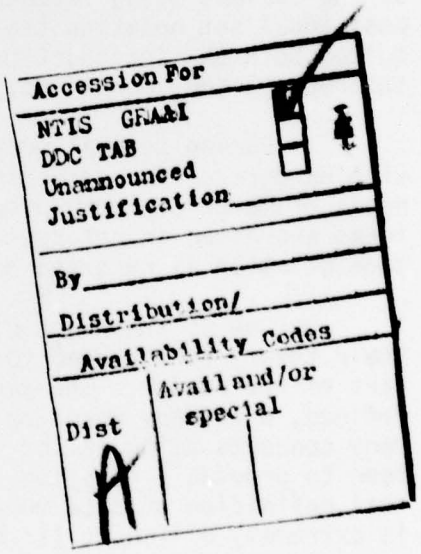
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper outlines a general philosophy and framework for the specification of a data model. Extracts from scenarios that have been developed for specification of several well-known data models (e.g., DBTG, relational, TDMS) are given and discussed. These extracts are preliminary and imcomplete, but they are used to illustrate the feasibility of the Data Model Processor Approach. The scenarios are intended for implementation of an augmented positional processor that will be able to support different data models, and, through transformation, serve in defining and testing new DBMS standards.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents



1.0 Introduction

The idea that data models can be equivalent to one another and the belief that the cost of (expensive) conversion between database management systems can be achieved automatically are not new. Indeed the concept of the universal language and of ease of translation across languages arose early in modern computing history. However, one of the most elusive parts is in finding a model that allows such translation– for it must be general enough to cover all such systems.

In searching for methods of data representation, data semantics, and consequently methods of translation, the authors started to work on a data model processor that could provide a framework for testing data models as well as allow development of a data model theory. This report provides a short history of the project, as well as stating some of the most recent concepts, and finally makes some suggestions for future development.

2.0 Data Model Definitions and Concepts

During the past five years, there has been a coordinated data model study group in the Department of Information Systems Management at the University of Maryland. The work of this group commenced when Hardgrave worked with Childs (ref. 1) in understanding the theory and applicability of Extended Set Theory. This led Hardgrave to starting an implementation of the concept using integer sets (ref. 2) and then in expanding this into positional set notation (ref. 3) for defining data models. The work required both the introduction of new implementation methods as well as some theoretical considerations.

During several working sessions in 1975, Rothnie and Sibley worked with Hardgrave in considering the concepts of data, data models, and the needs of large scale distributed database systems. This led to several new ideas expressed in reference 4, and this formed a basis for further work, some of which is reported here.

One of the major difficulties with writing about data models and their theory is the need for a concise definition and understanding on the part of the reader. The jargon of data management is notoriously poorly defined, with many words being used for the same (or similar) concepts and many concepts being called by the same word. In fact, one of our goals has been to provide a notation and framework that encourages precise mathematical definition of data modelling concepts. Unfortunately, such discussion is extremely difficult (if not impossible) to condense. This has led to several extremely long papers (ref. 5, 6, 7, and 8), and even these have not been independent.

Our recent work has taken three important steps:

i) The implementation, simulation, and improvement of a real positional set processor-PSP (ref. 7).

ii) The theoretical framework of data models, and the provision of an augmented PSP that can test and support several data models (ref. 8).

iii) The beginnings of work in the use of data model theory for the development of a better understanding of underlying concepts

that must be understood for future standardization of DBMS. Some extracts from previous reports are now included for completeness.

2.1 The Basic Concepts of the Positional Set Processor

The notion of the positional set is the basic concept of the model under consideration. We shall briefly review the material of as follows. The construction

$$S = [\ X_1{}^{P_1},\ X_2{}^{P_2},\ \ldots\ X_n{}^{P_n}\ ]$$

is a positional-set. The pairs may be called duplexes. We call P_i the positional identifier (position). The positional identifier may be "null" (denoted by #).

The component X_i we call the value of the element (or simply the element. The element can be atomic or the element value can be a positional set; thus allowing a form of recursiveness.

The value of X_i can be single-valued if

$$\forall_{i,j}\ [\ i \neq j \rightarrow P_i \neq P_j\ ]$$

or multi-valued if

$$\exists_{i,j}\ [\ i \neq j \rightarrow P_i = P_j]$$

that is, there exist several duplexes with the same positional identifiers.

If

$$\forall_i\ [\ P_i = \#\ ]$$

then the positional set

$$[\ X_1{}^{P_1},\ X_2{}^{P_2},\ \ldots\ X_n{}^{P_n}\ ]$$

is equivalent to the classical set

$$\{\ X_1,\ X_2,\ X_3,\ \ldots\ X_n\ \}.$$

The set

$$[\ X_1{}^{1},\ X_2{}^{2},\ \ldots\ X_n{}^{n}\ ]$$

is a classical sequence.

Further, we shall use primitive sets, which are classical sets of scalar values with different characteristics. The specific allowable constructs or sets of values are defined by the DBMS designer and are fixed in the data language associated with the model under consideration. In particular, we shall normally need to use primitive integer sets and real sets, character-strings of fixed and variable length, etc.

Some subset of the primitive sets we shall call the domain which represents the possible set of values (v). This primitive set defines the possible elementary values which are associated with one or more positional identifiers; several positional identifiers of one or several positional

sets can be associated with one domain (e.g., the domain TOWN may have PIDs 'RECEIVED FROM' and 'SENT TO'). The domain can be implicit or enumerated. An implicit domain is defined by giving the primitive set and the subsetting criteria for this subset. Typically, a PICTURE specifies the domain. Such criteria may be used by the DBMS as a domain integrity-constraint. For an enumerated domain, elements are listed explicitly. This list of values may also be used as an integrity-constraint.

Turning again to positional sets, we can see that data objects associated with them are built recursively. We define the depth of recursion of the atomic element to be zero. Then a positional set has recursion depth k if the maximum depth of recursion of its elements is k-1.

The instance of a positional set or tuple is, by definition, a positional set which conforms to the definitional constraints. The set of all instances of positional sets with the same schematic properties we may call the named positional set (e.g., a PERSONNEL positional set).

Although the sequence is a particular case of a positional set, there must be special specifications for this object, because it plays an important role in the model under consideration. In the same way that we distinguish a positional set according to its name and instance, we shall distinguish the name of a sequence from its instance. A sequence can be of fixed or variable length.

So, the positional set (sequence) may have an arbitrary level of recursion. It consists of duplexes. A duplex consists of a position identifier and an element. The values of an element can be the instances of a positional set of some type (sequence) or they may be atomic. The elementary value in this duplex can be single-valued or multi-valued. The conceptual data-base is the collection of positional sets with given characteristics.

The DBMS which uses such a conceptual model must have domain manipulation facilities, the facilities for manipulating named positional sets, their instances, duplexes, single-valued and multi-valued elements. Before we consider the data-manipulation facilities more precisely, we shall discuss the functional specifications of the objects.

2.2 Data Model Theory

In reference 4 the authors defined a data model to be:

- A collection of objects (or abstract entities);
- A data definition capability for those objects— presumably a stored structure describing objects; and
- A collection of operations that manipulate the objects by referencing names in the data definition structure.

In reference 5 the authors refined this definition somewhat to include specification of primitive sets from which other positional sets could be defined. Our approach is consistent with reference 4. That is, a data model must have specified as a minimum:

1. A collection of primitive sets.

2. A collection of objects (i.e., abstract entities). The form of each object must be given in Positional-Set-Notation (PSN). Further, the capability for naming an object must be specified with an indication of who (e.g., the end-user or the DBA) may name it.
3. A stored structure representing a data definition capability. This structure must also be given in PSN; it may be as complex as necessary.
4. A collection of operations that manipulate the objects (usually) by referencing the data definition.

As an example, we have given a data model definition for the relational model in reference 6.

3.0 Data Model Processor Concepts

The development of a Data Model Processor will be a significant step towards a general understanding of data models. This research involves several important parts: first, the development of a general framework for the specification of data models; secondly, the specification of each data model (e.g., DBTG, Relational, TDMS, IMS) within this framework; third, the specification of query languages for each data model. Using this approach, several query languages may be specified for a given data model. (The relational systems may be considered to form one data model which has several query languages, such as SEQUEL 2, QUEL, and QUERY-BY-EXAMPLE.) In the fourth part of the research, it is necessary to define a method which allows specification of mappings. Mappings across data-models must be possible, as well as mappings within one data model. Each of these topics is now discussed in some detail to show how far the research has currently progressed.

To demonstrate the Data Model Processor approach, we must give precise formulations for a number of existing data models. In many cases, this means augmenting (and sometimes altering) the original formulation. For example, the original relational model does not provide a mechanism for defining domains. Our mathematical approaches are not intended to be final; but they are necessary to illustrate this approach.

3.1 The Data Model Processor (DMP)

We see the framework as an on-line interactive system that provides several options which are designed to support the study of data models. The user of the DMP system would first encounter a "master menu" shown in Figure 3-1.

Each option is associated with a role shown in Figure 3-2. The person sitting at the terminal may play several different roles, but it is important to distinguish between them when using the system— they provide a means of controlling the model, its implementation, and use.

The Data-Model-Definer (DMD) has the most powerful role: indeed it is the DMD who delegates authority to all other roles during part of the data-model specification. The DMD specifies a data-model by:

```
DATA MODEL PROCESSOR:  23 JAN 1979

  SELECT ONE:

     M:  DEFINE NEW DATA MODEL     (DMD)
     I:  IMPLEMENT AN EXISTING DM  (DI)
     D:  DEFINE A NEW DATA BASE    (DBD)
     P:  POPULATE A DATA BASE      (DBP)
     R:  RETRIEVE,UPDATE, ETC.     (DM)
     Q:  DEFINE A QUERY LANGUAGE   (QLD)
     T:  DEFINE TRANSFORMATIONS    (DTD)
     H:  HELP (I.E. PRINT THIS MENU.)
     X:  EXIT
```

Figure 3-1: MASTER MENU

```
ROLES:

    DMD:  DATA MODEL DEFINER
    DI :  DBMS IMPLEMENTER
    DBD:  DATA-BASE DEFINER
    DBP:  DATA-BASE POPULATOR
    DM :  DATA MANIPULATOR
    QLD:  QUERY LANGUAGE DEFINER
    DTD:  DATA TRANSFORMATION DEFINER
```

Figure 3-2: ROLES

- Naming the concepts for the data-model;
- Defining the stored-data-definition (in PSN) for each concept;
- Defining the occurrence-structures (in PSN) for each concept;
- Defining the primitive-operations; and
- Specifying the source of each positional-set.

The meaning of "implementation" in this approach is not the traditional one. The DMD may specify that some sets are defined at implementation time: e.g., the set of real numbers usually depends on the hardware architecture of a specific computer. This, and any number of other sets, may be left to the "implementor" role.

The DBMS implementor (DI) of the data-model specifies these sets and does nothing else. This approach forces the DI to be perfectly faithful to the concepts set forth by the DMD. This is in marked contrast to the way DBMS have been implemented in the past. For example, in his original paper, Codd (reference 9), playing the DMD role, specified that relations would never have duplicate tuples; IBM's System-R group (reference 10), playing the DI role, allowed duplicate tuples, except when explicitly removed by the user. Such a condition could not occur under the DMP framework because it gives the DMD absolute control over implementation.

Once an implementation is available, a data-base may be defined under a new role (DBD), then populated (DBP), and finally manipulated (DM). The duties of the DBD and DBP roughly correspond to the duties of a data-base administrator (DBA). Since this term is used in a number of different contexts, it has been avoided in the definition of the DMP framework.

The DMP approach also separates the notion of a data-model from the notion of a query-language. The interface between the two is the collection of primitive operations. The DMD specifies the primitive operations; the QLD specifies the syntax of the query-language and a translation of the syntax to the primitive operations. In section 3.3, more is given on this.

The DMP approach should also provide for the specification of mappings or transformations (through the DTD). So far, the mechanism for doing this has not been developed.

3.2 Specification of Data Models

In this section, we discuss the specification of three data models: DBTG, Relational, and TDMS. Each specification is currently several pages long. Furthermore, the current specifications are preliminary and (in some cases) incomplete. However, portions of each specification are given here to illustrate how the data-model-processor would manage and manipulate these specifications.

3.2.1 Data Model Underlying Concepts

The concepts of the (original) DBTG model are:

- AREAS
- RECORDS
- SETS.

The concepts of the Relational model are:

- DOMAINS (Explicit and Implicit)
- RELATIONS.

Explicit domains are defined by enumerating the legal values. Implicit domains are defined by providing a defining predicate.

The concepts of the TDMS model are:

- COMPONENTS
- NODES
- INDEXES.

3.2.2 Stored Data Definitions

The Stored-Data-Definitions (SDD) for the DBTG, Relational, and TDMS models are given in Figures 3.2-1, 3.2-2, and 3.2-3 respectively.

The TEMPLATE command of DMP allows specification of the general form of a positional set. For example, the specification of RECORD-DEF in Figure 3.2-1 states that DECORD-DEF is a (mathematical) set of tuples. Each tuple has three position-identifiers: RECORD-NAME, AREA-NAME, and DATA-ITEM. The structures D(I) at position DATA-ITEM are themselves sets of tuples as specified in the subordinate TEMPLATE command.

Thus, in the DBTG model, each of the "concepts" (areas, records, and sets) have stored-data-definitions associated with them.

In our definition of the relational model, both domains and relations have stored-data-definitions associated with them, however, actual values may appear in XDOM-DEF making it similar in some ways to an occurrence structure.

3.2.3 Occurrence Structures

Figure 3.2-4 shows (some of) the occurrence structures for the DBTG model. The currency indicators are an important collection because the operations (e.g., FIND) depend on their values.

Figure 3.2-5 shows the occurrence structures for the Relational model. This consists of a set of pairs: relation-name and relation. The relation is itself a structure consisting of sets of tuples; the position-ids of which must conform to some restrictions set forth in the stored-data-definition. Here we will not discuss details needed for enforcing those restrictions since they are likely to change in our next version.

Figure 3.2-6 shows one occurrence structure for the TDMS model: the nodes. As discussed by Hardgrave in reference 11, the semantics of TDMS operations are made in terms of manipulations of the nodes in the tree. Another occurrence structure which may be included in the index structure of TDMS. There is considerable debate amongst us over the inclusion of index structures in the data model definitions. This is more controversial in the TDMS model than in others. It may not, in fact, be resolved for some time.

```
TEMPLATE AREA-DEF = {[N(I)@AREA-NAME,NO(I)@AREA-NO],...}
    WHERE
      N(I) IS-IN NAMES,
      NO(I) IS-IN AREA-NUMBERS;

TEMPLATE RECORD-DEF = {[N(I)@RECORD-NAME,A(I)@AREA-NAME,
                                D(I)@DATA-ITEM],...}
     WHERE
       N(I) IS-IN NAMES,
       A(I) IS-IN PROJ(AREA-DEF,AREA-NAME),
       TEMPLATE D(I) = {[AT(J)@ATTRIBUTE,P(J)@PICTURE],...}
         WHERE
           AT(J) IS-IN NAMES,
           P(J) IS-IN PICTURES;

TEMPLATE SET-DEF = {[N(I)@SET-NAME,O(I)@OWNER,S(I)@MEMBERS],...}
  WHERE
    N(I) IS-IN NAMES,
    O(I) IS-IN PROJ(RECORD-DEF,RECORD-NAME),
    TEMPLATE S(I) = {[M(J)@MEMBER,AM(J)@AUTO-MAN],...}
      WHERE
        M(J) IS-IN PROJ(RECORD-DEF,RECORD-NAME),
        AM(J) IS-IN {'AUTOMATIC','MANUAL'},*H;
```

Figure 3.2-1: DBTG STORED DATA DEFINITION

```
TEMPLATE XDOM-DEF = {(N(I)@NAME,Y(I)@EXTENDABLE,V(I)@VALUES),...}
    WHERE N(I) IS-IN NAMES,
          Y(I) IS-IN ('YES','NO'),
          TEMPLATE V(I) = {X(J),...}
                          WHERE X(J) IS-IN ATOMS;

TEMPLATE IDOM-DEF = {(M(I)@NAME,L(I)@LOW,H(I)@HIGH),...}
    WHERE M(I) IS-IN NAMES,
          L(I) IS-IN ATOMS,
          H(I) IS-IN ATOMS,
          L(I) < H(I);

LET DOMS = PROJ(XDOM-DEF,NAME) UNION PROJ(IDOM-DEF,NAME);

TEMPLATE REL-DEF = {(RN(I)@NAME,SR(I)@A-D-PAIRS),...}
    WHERE RN(I) IS-IN NAMES,
          TEMPLATE SR(I) = {(A(J)@ATTRIBUTE,K(J)@KEY-PART,
                             D(J)@DOMAIN),...}
          WHERE A(J) IS-IN NAMES,
                K(J) IS-IN ('YES','NO'),
                D(J) IS-IN DOMS;
```

Figure 3.2-2: RELATIONAL STORED DATA DEFINITION

```
ENTER P-SETS;
  TEMPLATE COMP-DEF =
     {( D(I)@NUMBER, N(I)@NAME,T(I)@TYPE,K(I)@KEY,Q(I)@PICTURE,
        P(I)@PARENT),...}
     WHERE
        D(I) IS-IN COMP-NUMBERS,
        N(I) IS-IN NAMES,
        T(I) IS-IN TYPES,
        K(I) IS-IN {'K','NK'}
        P(I) IS-IN COMP-NUMBERS,
        Q(I) IS-IN PICTURES;
```

Figure 3.2-3: TDMS STORED DATA DEFINITION

```
TEMPLATE RECORD-OCC = {[K(I)@DBK,N(I)@NAME,D(I)@DATA-ITEMS],...}
  WHERE
    K(I) IS-IN DBK,
    N(I) IS-IN PROJ(RECORD-DEF,RECORD-NAME),
    TEMPLATE D(I) = [V(J)@P(J),...]
      WHERE
        P(J) IS-IN ATTR(N(I)),
        V(J) IS-IN PICT(N(I),P(J));


TEMPLATE SET-LINKS = {[N(I)@SET-NAME, S(I)@LINKS],...}
  WHERE
    N(I) IS-IN PROJ(SET-DEF,SET-NAME),
    TEMPLATE S(I) = {[K1(J)@CURRENT,K2(J)@NEXT,K3(J)@PRIOR,
                      K4(J)@OWNER],...}
      WHERE
        K1(J) IS-IN DBK,
        K2(J) IS-IN DBK,
        K3(J) IS-IN DBK,
        K4(J) IS-IN DBK;


TEMPLATE CURRENCY-INDICATORS = {[N(I)@INDICATOR,T(I)@TYPE,V(I)@DBK],...}
  WHERE
    N(I) IS-IN CURRENCY-CATEGORIES,
    T(I) IS-IN {'RUN-UNIT','RECORD','SET','AREA'},
    V(I) IS-IN DBK;
```

Figure 3.2-4: DBTG OCCURRENCE STRUCTURES

```
TEMPLATE REL-OCC = {[N(I)@NAME,R(I)@RELATION),...}
  WHERE
    N(I) IS-IN NAMES,
    TEMPLATE R(I) = {[V(J,K)@P(J),...J...],...K...}
       WHERE
         P(J) IS-IN ATTF(N(I)),
         INCLUDE(V(J,K),DOMF(N(I),P(J))) IS TRUE;
```


Figure 3.2-5: RELATIONAL OCCURRENCE STRUCTURE

```
TEMPLATE NODES-OCC  =
   {[N(I)@NODE-ID,P(I)@RGID,U(I)@UP,D(I)@DOWN,R(I)@RIGHT,
     V(I)@VALUES],...}

   WHERE
     N(I) IS-IN NODES,
     R(I) IS-IN COMP-NUMBERS,
     U(I) IS-IN NODES,
     D(I) IS-IN NODES,
     R(I) IS-IN NODES,
     TEMPLATE(V(I)) = {[C(J)@COMP-NUMBER,X(J)@VALUE],...}
         WHERE
           C(J) IS-IN COMP-NUMBERS
           X(J) IS-IN ATOMS
```

Figure 3.2-6: TDMS OCCURRENCE STRUCTURES

3.3 Query Languages

One important aspect of this work is the development of a framework that encompasses query languages as well as data models. A recent study under the contract (reference 11) shows that one query language syntax may have several semantic interpretations (called query-philosophies) for a single data model. The query language module of the Data Model Processor should be general enough to handle this case. The work described here is still in an early stage of development.

The query language module of the Data Model Processor requires the following material for each query language:

- A GRAMMAR TABLE, and
- A SEMANTIC TABLE for each query-philosophy.

The grammatical table (currently) consists of a collection of rules. Each rule has a rule number, a match condition, a rewrite string, and a semantic symbol. Figure 3.3-1 shows a GRAMMAR TABLE for the TDMS query syntax.

Metasymbols are enclosed in angle brackets: <,>. These are similar to metasymbols in BNG grammars except they have both syntactic and semantic parts. The syntactic part is matched and rewritten in the normal way; however, the semantic part is a name assigned to the positional-set that is the current "mathematical value" of the expression. As the syntactic grammar is parsed, the positional-set is "calculated" in parrallel. This is done by referencing the appropriate semantic table using the SEMSYM field. Figures 3.3-2 and 3.3-3 show the semantic tables for the form TDMS query-*philosophies*.

DMP also provides a mechanism for testing queries after a data model and query language are defined. Of course a test data-base must have been defined and populated as well. Figures 3.3-4, 3.3-5, and 3.3-6 show the processing of a query using this approach. The query and sample data base are taken from reference 11.

4.0 Continuing Work

4.1 DMP at NBS

The Data Model Processor is currently being specified and a prototype will be implemented on The Experimental Computer Facility at the U.S. National Bureau of Standards. Most of the work on DMP will be supported by NBS under their research project on abstract data models.

4.2 Mappings

The definition and processing of mappings may be the most important aspect of the DMP design. A good mapping facility would allow DMP to simulate ANSI/SPARC architecture and also provide a foundation for data-base conversion and schema transformations. So far, very little work has been done on this part of the DMP design, though the effort ultimately requires this in order to be complete.

```
SELECT OPTION: G
ENTER NAME OF MODEL: TDMS
ENTER NAME OF QUERY LANGUAGE: QL200
ENTER NAMES OF QUERY PHILOSOPHIES:
     SET-BARS, TREE-BARS/T,TREE-BARS/N,ROOT-BARS,$END

ENTER GRAMMATICAL CONSTRUCTS:

RULE#  MATCH                          REWRITE                              SEMSYM
-----  -----                          -------                              ------
1      EQ!NE!GE!GT!LT!LE              <OP/$1>                              NULL
2      <Z(1)><OP/#X><Z(2)>            <ATT/Z(1)><OP/#X><VAL/Z(2)>          NULL
3      <ATT/#X><OP/#Y><VAL/#Z>        <EP/S(K)>                            EP
4      <EP/S(I)>                      <COND/S(I)>                          NULL
5      <COND/S(I)>AND<COND/S(J)>      <COND/S(K)>                          C1
6      <COND/S(I)>OR <COND/S(J)>      <COND/S(K)>                          C2
7      NOT <COND/S(I)>                <COND/S(K)>                          C3
8      ( <COND/S(I)> )                <COND/S(I)>                          NULL
9      <Z(1)> HAS <COND/S(I)>         <COND/S(K)>                          HC
10     QUALIFY WHERE <COND/S(I)>      <QS/Q>                               QS
11     FAMILY IS <Z(1)>               <FM/T>                               FM
$END
```

Figure 3.3-1: GRAMMAR TABLE FOR TDMS QUERIES

```
ENTER SEMANTIC TABLE FOR SET-BARS:

CONSTRUCT   OUTPUT   SET-BARS
---------   ------   --------
QS          Q        S(I)
EP          S(K)     QUALIFY(SELECT(<ATT>,<OP>,<VAL>)
C1          S(K)     INTER(S(I),S(J))
C2          S(K)     UNION(S(I),S(J))
C3          S(K)     RLCOMP(UNIVERSE,S(I))
HC          S(K)     QUALIFY(ADJUST(<ATT>,S(I)))
$END
ENTER UNIVERSE: UNION-ALL TYPE(T) FOR ALL T;


ENTER SEMANTIC TABLE FOR TREE-BARS/T:

CONSTRUCT   OUTPUT   TREE-BARS/T
---------   ------   -----------
QS          Q        QUALIFY(S(I))
EP          S(K)     ADJUST(SELECT(<ATT>,<OP>,<VAL>),T)
C1          S(K)     INTER(S(I),S(J))
C2          S(K)     UNION(S(I),S(J))
C3          S(K)     RLCOMP(TYPE(T),S(I))
HC          S(K)     ADJUST(ADJUST(S(I),<ATT>),T)
FM          T        FAMILY(<ATT>)
$END
```

Figure 3.3-2: SEMANTIC TABLES FOR TDMS-LIKE QUERIES (A)

```
ENTER SEMANTIC TABLE FOR TREE-BARS/N:

CONSTRUCT   OUTPUT   TREE-BARS/N
---------   ------   -----------
QS          Q        QUALIFY(S(I))
EP          S(K)     SELECT(<ATT>,<OP>,<VAL>)
C1          M        LOW(S(I),S(J));
            S(K)     INTER(ADJUST(S(I),M),ADJUST(S(J),M))
C2          M        LOW(S(I),S(J));
            S(K)     UNION(ADJUST(S(I),M),ADJUST(S(J),M)
C3          S(K)     RLCOMP(TYPE(RG(S(I)),S(I))
HC          S(K)     ADJUST(S(I),<ATT>)
$END

ENTER SEMANTIC TABLE FOR ROOT-BARS:

CONSTRUCT   OUTPUT   ROOT-BARS
---------   ------   ---------
QS          Q        QUALIFY(S(I))
EP          S(K)     ADJUST(SELECT(<ATT>,<OP>,<VAL>),1)
C1          S(K)     INTER(S(I),S(J))
C2          S(K)     UNION(S(I),S(J))
C3          S(K)     RLCOMP(TYPE(1),S(I))
HC          S(K)     S(I)
$END
```

Figure 3.3-3: SEMANTIC TABLES FOR TDMS-LIKE QUERIES (B)

```
SELECT OPTION: R
ENTER NAME OF DATA MODEL OR QUERY LANGUAGE: QL200
ENTER NAME OF DATA BASE: COMPUTING EQUIPMENT
ENTER NAME OF QUERY PHILOSOPHY: SET-BARS

ENTER QUERY: QUALIFY WHERE (( C11 EQ PHYSICS OR C12 GT 50) AND NOT
               ((C21 HAS C31 EQ PRINTER) AND (C21 HAS C32 EQ PERTEC);

ENTER NAME OF QUERY PHILOSOPHY: SET-BARS

PRIMITIVE FUNCTION STREAM IS:

CON   OUTPUT   FUNCTIONS                                  RESULT
---   ------   ---------                                  ------
EP    S(1)     QUALIFY(SELECT(C11,EQ,PHYSICS))            {1,4,5,6,12,13,14,15}
EP    S(2)     QUALIFY(SELECT(C12,GT,50)                  {3,9,10,11,20,21}
EP    S(3)     QUALIFY(SELECT(C31,EQ,PRINTER)             {1,2,5,8,12,16}
EP    S(4)     QUALIFY(SELECT(C32,EQ,PERTEC)              {1,5,13}
HC    S(5)     QUALIFY(ADJUST(C21,S(3)))                  {1,2,5,8,12,...19}
HC    S(6)     QUALIFY(ADJUST(C21,S(4)))                  {1,5,12,13,14,15}
C2    S(7)     UNION(S(1),S(2))                           {1,3,4,5,6,9,...,15,
                                                           20,21}
C1    S(8)     INTER(S(5),S(6))                           {1,5,12,...,15}
C3    S(9)     RLCOMP(UNIVERSE,S(8))                      {2,3,4,6,8,...,11,
                                                           16,...21}
C1    S(10)    INTER(S(7),S(9))                           {3,4,6,9,10,11,20,21}
QS    S(11)    S(10)                                      {3,4,6,9,10,11,20,21}
```

Figure 3.3-4: PROCESSING A QUERY (A)

```
ENTER QUERY PHILOSOPHY: TREE-BARS/T

PRIMITIVE FUNCTION STREAM IS:

CON   OUTPUT   FUNCTIONS                                        RESULT
---   ------   ---------                                        ------
EP    S(1)     ADJUST(SELECT(C11,EQ,PHYSICS),C31)               {12,13}
EP    S(2)     ADJUST(SELECT(C12,GT,50),C31)                    {20}
EP    S(3)     ADJUST(SELECT(C31,EQ,PRINTER),C31)               {12,16}
EP    S(4)     ADJUST(SELECT(C32,EQ,PERTEC),C31)                {13}
HC    S(5)     ADJUST(ADJUST(S(3),C21),C31)                     {12,13,16,17}
HC    S(6)     ADJUST(ADJUST(S(4),C21),C31)                     {12,13}
C2    S(7)     UNION(S(1),S(2))                                 {12,13,20}
C1    S(8)     INTER(S(5),S(6))                                 {12,13}
C3    S(9)     RLCOMP(TYPE(C31),S(8))                           {16,17,20}
C1    S(10)    INTER(S(7),S(9))                                 {20}
QS    S(11)    QUALIFY(S(10))                                   {3,11,20}

ENTER QUERY PHILOSOPHY: TREE-BARS/N

PRIMITIVE FUNCTION STREAM IS:

CON   OUTPUT   FUNCTIONS                                        RESULT
---   ------   ---------                                        ------
EP    S(1)     SELECT(C11,EQ,PHYSICS)                           {1}
EP    S(2)     SELECT(C12,GT,50)                                {3}
EP    S(3)     SELECT(C31,EQ,PRINTER)                           {12,16}
EP    S(4)     SELECT(C32,EQ,PERTEC)                            {13}
HC    S(5)     ADJUST(S(3),C21)                                 {5,8}
HC    S(6)     ADJUST(S(4),C21)                                 {5}
C2    M        C11
      S(7)     UNION(ADJUST(S(1),C11),ADJUST(S(2),C21))         {1,3}
C1    M        C21
      S(8)     INTER(ADJUST(S(5),C21),ADJUST(S(6),C21))         {5}
C3    S(9)     RLCOMP(TYPE(RG(S(8))),S(8))                      {8,11}
C1    M        C21
      S(10)    INTER(ADJUST(S(7),C21),ADJUST(S(9),C21))         {11}
QS    S(11)    QUALIFY(S(10))                                   {3,11,20,21}
```

Figure 3.3-5: PROCESSING A QUERY (B)

```
ENTER QUERY PHILOSOPHY: ROOT-BARS

PRIMITIVE FUNCTION STREAM IS:

CON   OUTPUT  FUNCTIONS                                    RESULT
---   ------  ---------                                    ------
EP    S(1)    ADJUST(SELECT(C11,EQ,PHYSICS)                {1}
EP    S(2)    ADJUST(SELECT(C12,GT,50)                     {3}
EP    S(3)    ADJUST(SELECT(C31,EQ,PRINTER))               {1,2}
EP    S(4)    ADJUST(SELECT(C32,EQ,PERTEC),C00)            {1}
HC    S(5)    S(3)                                         {1,2}
HC    S(6)    S(4)                                         {1}
C2    S(7)    UNION(S(1),S(2))                             {1,3}
C1    S(8)    INTER(S(5),S(6))                             {1}
C3    S(9)    RLCOMP(TYPE(C00),S(8))                       {2,3}
C1    S(10)   INTER(S(7),S(9))                             {3}
QS    S(11)   QUALIFY(S(10))                               {3,9,10,11,18,...21}

ENTER QUERY PHILOSOPHY: $END
ENTER QUERY: $END

END OF QUERY SESSION.
```

Figure 3.3-6: PROCESSING A QUERY (C)

4.3 Standards

We are partly interested in DMP because of its implications for data-base standards. This also makes it of interest to NBS, partly because:

(1) Congress and OMB expect data-base standards for DDL, DML and query facilities for Federal procurements to be available by 1985.

(2) The only data-model that is seriously being considered for standardization is DBTG.

(3) A DBTG standard that ignores other data models may mislead Federal users.

DMP would provide a general framework for standardization. First, a precise definition of a data model could be developed and (eventually) accepted as a means of work by standardization committees. Moreover, this would not preclude the definition and eventual standardization of other data models. Multiple query-languages may be defined (and standardized) as required. The mapping facility may also be used to determine definitively whether two query languages (or data models) are equivalent.

4.4 Terminology

One of the most important results of DMP development will be the precise definition of some current terms from data-base technology. Different authors use terms such as "attribute," "mapping," "data-base," and "data-model" with similar but confusingly different meanings. Newcomers to the field are often perplexed by the plethora of pseudo-defined terms. The DMP development requires precise definitions for many of these and other terms. Thus we hope to introduce a new precision in definition to this field.

5.0 Bibliography

1. Childs, D.L. Feasibility of a set-theoretic data structure: A general structure based on a reconstructed definition of relation. Proc. IFIP Congress 1968, North Holland Press, Amsterdam 1968, 420-430.

2. Hardgrave, W.T. Technique for implementing a set processor. ACM SIGPLAN Notices, Vol. II, 1976 Special Issue, In FDT 8,2 (1976), 86-94.

3. Hardgrave, W.T. The relational model: A reformulation of some mathematical aspects. IFSM T.R. No. 25, Version 2, Dept. of Information Systems Management, University of Maryland, College Park, MD, 20742, March 1978.

4. Rothnie, J.B., and Hardgrave, W.T. Data model theory: A beginning. Proc. of the Fifth Texas Conference on Computing Systems, October 1976 (Note: only an abstract appears due to mail delay). Also available as IFSM T.R. No. 10, Dept. of Information Systems Management, University of Maryland, College Park, MD, 20742, September 1976.

5. Sibley, E.H., and Hardgrave, W.T. Data model theory and positional set processing. In Data Models and Database Systems, Proc. of Joint US-USSR Seminar, Moscow, November 14-23, 1977, Institute for Computing Science and Computer Applications, Univ. of Texas at Austin, 5-71.

6. Sibley, E.H., Hardgrave, W.T., Kogalovsky, M.R., and Makalsky, K.I. A conceptual model to support multi-model external views. Proc. of Joint US-USSR Symposium on Data Base, Austin, October 25-27, 1979. To be published by Institute for Computing Science and Computer Applications, Univ. of Texas at Austin.

7. Hardgrave, W.T., Sibley, E.H., Kogalovsky, M.R., and Kogutovsky, V.V. Problems of positional and integer set processor implementations. Proc. of Joint US-USSR Symposium on Data Base, Austin, October 25-27, 1979. To be published by Institute for Computing Science and Computer Applications, Univ. of Texas at Austin.

8. Sibley, E.H., and Hardgrave, W.T. The positional set processor as a model for three schema architectures. IFSM T.R. No. 38, Dept. of Information Systems Management, University of Maryland, College Park, MD, 20742, January 1979.

9. Codd, E.F. A relation model of data for large shared data banks. Comm. of the ACM 13,6 (June 1970), 377-387.

10. Astrahan, M.M., Chamberlin, D.D., et al. System-R, a relational approach to data base management. TODS 1,2 (July 1976), 97-137.

11. Hardgrave, W.T. Ambiguity in processing boolean queries on TDMS tree structures: A study of four philosophies. IFSM T.R. No. 35, Version 2, Dept. of Information Systems Management, University of Maryland, College Park, MD, 20742, July 1979.